LA-UR-00-6049
Approved for public release;
distribution is unlimited.

Title:	CartaBlanca– A Pure-Java, Component-based Systems Simulation Tool for Coupled Non-linear Physics on Unstructured Grids
Author(s):	*W. B. VanderHeyden* *E. D. Dendy* *N. T. Padial-Collins*
Submitted to:	*Joint ACM Java Grande - ISCOPE 2001 Conference* *Stanford, California* *June 2-4, 2001*

Los Alamos
NATIONAL LABORATORY

CartaBlanca– A Pure-Java, Component-based Systems Simulation Tool for Coupled Non-linear Physics on Unstructured Grids©

W. B. VanderHeyden
Los Alamos National Laboratory
Theoretical Division and
Los Alamos Computer Science Institute
Los Alamos, NM 87545
wbv@lanl.gov

E. D. Dendy
Los Alamos National Laboratory
Theoretical Division and
Los Alamos Computer Science Institute
Los Alamos, NM 87545
dendy@lanl.gov

N. T. Padial-Collins
Los Alamos National Laboratory
Theoretical Division and
Los Alamos Computer Science Institute
Los Alamos, NM 87545
nelylanl@lanl.gov

ABSTRACT
This paper describes a component-based non-linear physical system simulation prototyping package written entirely in Java using object-oriented design to provide scientists and engineers a "developer-friendly" software environment for large-scale computational method and physical model development. The software design centers on the Jacobian-Free Newton-Krylov solution method surrounding a finite-volume treatment of conservation equations. This enables a clean component-based implementation. We first provide motivation for the development of the software and then describe software structure. Discussion of software structure includes a description of the use of Java's built-in thread facility that enables data-parallel, shared-memory computations on a wide variety of unstructured grids with triangular, quadrilateral, tetrahedral and hexahedral elements. We also discuss the use of Java's inheritance mechanism in the construction of a hierarchy of physics-systems objects and linear and non-linear solver objects that simplify development and foster software re-use. As a compliment to the discussion of these object hierarchies, we provide a brief review of the Jacobian-Free Newton-Krylov nonlinear system solution method and discuss how it fits into our design. Following this, we show results from preliminary calculations and then discuss future plans including the extension of the software to distributed memory computer systems.

Categories and Subject Descriptors
D.1.3 [Software] Programming Techniques—*Concurrent programming;* D.1.5 [Software]: Programming Techniques—*Object-Oriented Programming;* D.2.6 [Software]: Software Engineering—*Programming Environment;* D.2 13 [Software]: Software Engineering—Reusable Software

General Terms
Design, Documentation

Keywords
Java, object oriented, components, solver, Newton, Krylov, Jacobian, threads, parallel, physics

JavaGrande/ISCOPE 2001 Stanford, California USA

1. INTRODUCTION
Specialized simulation software for non-linear physical systems is one of the central research products from many of the programs here at Los Alamos National Laboratory (LANL) as well as other similar institutions. These systems are often 3-dimensional and are solved on unstructured computational grids. Thus, these software systems can be quite complex. Very often, these projects involve the modification of existing software to produce new capabilities. Furthermore, program goals change frequently as funding priorities change or as research developments lead to new branches of investigation. As such, application development is often, by far, the bottleneck on these projects and there is, therefore, substantial incentive for software and software environments that are "developer-friendly"—easy for scientific and engineering software developers to modify and extend. Fortunately, there are two technology trends of recent years that we believe can be used to address this need in a coordinated fashion.

First, object-oriented and component-based software has made an enormous impact in the commercial software arena for telecommunications and e-Business. Enhancement of software developer productivity has come from a variety of sources including new languages such as Java which support object orientation and component software and a host of software developer productivity tools including graphical design applications, graphical debuggers, etc., that work with these languages. At the same time, significant advances have occurred recently in non-linear systems solutions methods. In particular, Jacobian-Free Newton-Krylov (JFNK), [3], methods have been employed and extended to provide robust and flexible solution methods for a wide variety of coupled non-linear physics

simulation capabilities. In addition, the combination of JFNK and the finite-volume technique, [7], fits well into an object-oriented or component-based software environment.

Thus, the goal of the CartaBlanca project is to produce a modern flexible software environment for prototyping physical models, discretization schemes and solution methods for non-linear physics problems on unstructured grids. CartaBlanca employs an object-oriented, component-based design using the Java programming language and takes advantage of appropriate available software engineering tools. In order to obtain a realistically useful software tool, CartaBlanca is written so that it can be used on a variety of unstructured grids in 2 and 3 dimensions. CartaBlanca is able to accept a variety of mesh elements including triangles and quadrilaterals in 2D and tetrahedral and hexahedra in 3D. CartaBlanca employs the finite-volume discretization method, [7], to provide for wide-ranging flexibility with regard to both meshes and physical systems. Finally, CartaBlanca is written in a data-parallel fashion to provide a test bed for parallel algorithm development.

1.1. Status

We currently have a great deal of infrastructure for CartaBlanca in place. We now have a fully 3-dimensional finite-volume simulation capability that can solve a variety of physics problems on unstructured meshes employing shared-memory parallelism using Java's built-in thread facility. The following is a list of features currently available

- Accepts unstructured grids in 2 and 3 dimensions and with triangular, tetrahedral, quadrilateral and hexahedral elements.
- Accepts Metis, [13], generated mesh partition files; computes in parallel on Metis sub domains using Java's built-in thread facility.
- Has a Graphical User Interface (GUI) based input facility.
- Abstract classes for state, physics and solvers objects impose a uniform structure for developers, enforced by the compiler.
- Pre-conditioned Conjugate gradient and GMRES Krylov linear solvers interact seamlessly with physics objects.
- Non-linear quasi-Newton solver is wrapped around linear Krylov solvers to provide a JFNK solver.
- Physics objects for high accuracy scalar advection, heat transfer and incompressible flow currently available.
- Automatic generation of Tecplot (from Amtec Engineering) graphics files for arbitrary physical systems.
- Embedded software-testing facility based on the JUnit framework, [12].
- Direct remote access to CVS revision control server is enabled for efficient team code development.

1.2. Java Performance

There have been numerous studies on the use of Java for high performance computing. The recent article by Schatzman and Donehower, [21], provides a useful discussion of the potential pitfalls involved and tips on how to program in Java to try to achieve performance comparable to that obtainable using C, C++ and Fortran. Despite the fact that numerical applications in Java are, at present, typically slower than equivalent applications written in these languages, it is our opinion that the important benefits from Java's strong typing, clean design, object-orientation and compatibility with so many useful commercial software productivity tools make it a serious alternative language for intensive scientific and engineering computing applications, even today. Also, as the Java language specification and Java interpreters and compilers improve we should see even greater benefits. Particularly encouraging are the Java language changes contemplated by the Java Grande Forum, [11].

1.3. Related Efforts

CartaBlanca builds on some important existing software programs here at LANL. CFDLIB, [14], a Fortran program, is an outgrowth of the Caveat program, [1], which provides developers with a flexible finite-volume multiphase flow simulation capability. The Telluride program, [23], here at LANL provides a multi-material simulation capability with interface tracking on unstructured grids. Telluride is written in Fortran 90 and makes extensive use of the Fortran 90 module concept. The CHAD program, [18], here at Los Alamos is a flexible node-based finite-volume simulation program for flow simulation on unstructured grids. CHAD also uses Fortran 90 and accommodates hybrid grids using an edge-based connectivity data structure, [22]. Finally, Kokopelli, is a C++ program for interfacial and polymer flow problems. The authors have had extensive experience with each one of these programs. The design and implementation of CartaBlanca builds on the lessons learned with each of these. Another LANL software effort worth mentioning here is the POOMA project, [17] , which used C++ and advanced object oriented programming techniques to produce a flexible scientific problem-solving environment. The POOMA experience, [17] provides useful insights into the use of object orientation in scientific computation.

It is also worthwhile to note two relevant examples from outside our laboratory. First, Hauser, et. Al., [10], have reported on their effort to produce an object-oriented, pure-Java simulation code for aerospace applications. They employ a multi-block structured grid computation scheme and use of Java's thread and RMI facilities for parallelization and to enable remote interaction between a graphical user interface and the numerical application.

Another effort worth noting here is the work of Hatakeyama, et. Al., [9], who describe an object-oriented paradigm for flow simulation software in which the object-oriented concept is used at the computational node level to produce flexible abstractions. They demonstrate their concepts with a C++, structured grid simulation of a wind tunnel with a test object and show that they can easily insert and extract arbitrary-shaped flow obstacles.

1.4. Outline

In Section 2 we provide an overview the finite-volume method for discretization of conservation equations. In Section 3 we describe the major features of the Jacobian-Free Newton-Krylov solution method. We then proceed in Section 4 to give an overview of the CartaBlanca software packages. We follow this in Section 5 with some basic results. Finally, in Section 6 we provide a discussion of conclusions and future plans.

2. FINITE VOLUME METHOD

CartaBlanca is based on the finite volume method, [7], for conservation equations. More specifically, CartaBlanca adopts the node-based version of this scheme with edge-based connectivity, [18], [22]. We provide here a very simplified outline of the method. For an arbitrary control volume V with

bounding surface A the generic conservation statement is of the form

$$\frac{d}{dt}\int_V qdV + \oint_A \vec{f}\cdot\vec{n}dS + \int_V sdV = 0, \qquad (1)$$

where q is the density of some conserved quantity such as mass, momentum or energy, $\vec{f}$ is the local flux of this conserved quantity due to a variety of mechanisms, $\vec{n}$ is an outward normal vector defined on the surface of the control volume, and s is a generalized source density. The first and third integrals in Equation (1) are over the entire space of the control volume; the second integral is over the surface of the control volume. The derivative on the first integral quantity in Equation (1) is with respect to time. For numerical computations, Equation (1) is discretized on a computational grid. On such a grid, conservation nodes are connected by edges as shown in Figure 1.

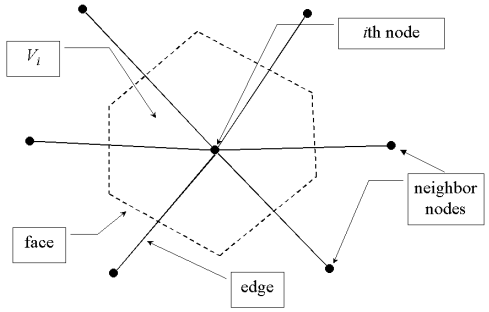

Figure 1. Control volume for ith node.

Each node is associated with a polyhedral control volume, V_i, as depicted in Figure 1. For each node, the averaged value of the conserved density is defined as

$$q_i \equiv \frac{1}{V_i}\int_{V_i} qdV. \qquad (2)$$

The quantities q_i are, typically, the state variables for the numerical simulation. Similarly, the average source over each control volume is

$$s_i \equiv \frac{1}{V_i}\int_{V_i} sdV. \qquad (3)$$

Let $\vec{f}_e$ be the average flux on the control volume face associated with edge e. Then, if we integrate the Equation (1) over a time step, Δt, using, for example, a first-order difference approximation for the time derivative, we obtain the discretized form of the conservation equation

$$q_i^{n+1}V_i^{n+1} - q_i^n V_i^n + \Delta t\left\{\sum_{edges}\vec{f}_e\cdot\vec{n}_e A_e + s_i V_i\right\} = 0, \qquad (4)$$

where the superscripts n and $n+1$ denote the present and future time levels. Of course, the fluxes and source terms are generally functions of space, time and the state variables, q_i. Thus, the set of discretized conservation equations for all nodes and all types of conservation quantities forms a nonlinear algebraic system. The physics for a given application lies in the definition of the fluxes and sources in Equation (4). The aim of CartaBlanca is to provide scientists and engineers a friendly environment using object-oriented, component-based Java for the implementation of physics and the solution of the resulting coupled nonlinear conservation equations.

3. JACOBIAN-FREE NEWTON-KRYLOV METHOD

We may write the set of conservation equations in the compact, abstract form

$$F_i\left(q^{n+1}\right) = 0 \qquad (5)$$

where F_i denotes the left hand side of Equation (4) and q^{n+1} denotes the entire set of state variables at the advanced time. The quantity F_i is called the residual function. The system represented by Equation (5) is, in general, nonlinear. We employ the Jacobian-Free Newton-Krylov method, [3], in CartaBlanca to solve these systems. We provide here a brief outline in order to motivate our discussion of the software design. Newton's method for a nonlinear system begins with an initial guess of the solution, $q_j^{n+1(0)}$, where the superscript in parenthesis denotes the iterate level. This is, typically, the solution from time level *n*. Newton's method then proceeds through a series of iterations involving the solution of a sequence of linear systems

$$J_{ij}\left(q^{n+1(k)}\right)d_j^k = -F_i\left(q^{n+1(k)}\right), \qquad (6)$$

along with the update

$$q^{n+1(k+1)} = q^{n+1(k)} + d^k, \qquad (7)$$

where there is an implied summation in Equation (6) on the repeated index, j. The goal, of course, is to proceed until we find the solution to Equation (5). The matrix quantity, J_{ij}, is the Jacobian matrix defined as

$$J_{ij}(q) = \frac{\partial F_i(q)}{\partial q_j}. \qquad (8)$$

Explicit formation of the Jacobian matrix is typically a very expensive computation. Fortunately, the JFNK method takes advantage of the fact that Krylov linear solution methods require only the evaluation of matrix-vector products, Jv (where v is a Krylov vector), and not the matrix J by itself, [3]. Furthermore, matrix-vector products can be approximated numerically using a directional difference formula,

$$Jv \approx \frac{F(q+ev) - F(q)}{e}, \qquad (9)$$

where e is some small scalar perturbation parameter, [3]. This approximation allows us to structure CartaBlanca in such a way that the physics developer can focus on providing residual functions inside physics objects or components. Using the abstraction embodied in Equation (5) we have genericized the rest of the infrastructure for solving and processing physics problems so that developers can work simultaneously on a variety of different problems using the same software.

4. SOFTWARE

CartaBlanca is composed, at present, of eleven separate packages. Each of these packages contain classes that perform distinct functions. We have tried to design these classes to serve, as much as possible, as software components that can be interchanged in a "plug and play" mode by developers. We have also tried to write the utility classes in such a way that many developers need not concern themselves with the parallel nature of the computation.

In the following we describe each of these packages and the classes they contain. We also describe the interactions and associations between the classes in the different packages. We choose here to start the discussion with the mesh and input packages. These are low-level packages in the sense that they are used by many other packages and make sparing use of other packages. We then work our way up through the remaining packages of increasing complexity until we finally describe the main package, which contains the main methods. Before proceeding to the discussion of the CartaBlanca software packages, we start by commenting on our general design approach and on our software engineering methods.

4.1. Approach

Our approach to the design of CartaBlanca includes the following general guiding principles. First, we have endeavored to make use of object-orientation at the highest levels from a physical point of view. Thus our objects are things like physical systems that exist over the entire sections of the computational domain or grid, rather than at the individual nodes. This choice was made based on the idea that this would yield higher numerical performance by avoiding excessive overhead at the node level and would also represent a smoother transition into object-oriented programming from the point of view of procedural scientific legacy codes. Nevertheless, this approach has allowed us to make substantial use of Java and its object-oriented features as will be seen below.

Another principle we have employed is to make the top levels of the program as generic as possible so that the developer can plug physics into the appropriate program locations and then have the rest of the program able to immediately interact. This was accomplished, in part, by the use of abstract classes, which provide virtual functionality and interfaces for things such as physics and solver objects. This imposes a certain structure on the derived classes that a developer provides.

4.2. Software Engineering

Because of the significant research content of our project, our approach to team programming follows the lightweight processes advocated in the recent article by Fowler, [8]. Iterative programming and component development has, for example, been very useful. The use of team coding has also proved helpful.

In order to foster the team software approach, we have incorporated the JUnit, [12], testing facility into CartaBlanca. This has been useful in that any developer can perform tests easily on their local computing platform to make sure his modifications have not corrupted the software. This is in contrast to a situation in which software testing is performed using specialized software available only on a certain computing platform.

We have found it very helpful to use a common integrated development environment (IDE) for our software development. We are currently using JBuilder 4.0 Professional by Borland Technologies. JBuilder gives us an identical programming environment on our Windows NT and Solaris workstations. JBuilder is also available for LINUX operating systems. The JBuilder environment, conveniently, recognizes the JavaDoc @todo functionality. We use this feature as a simple issues tracking mechanism.

In addition to the JBuilder IDE, we use the GNU CVS revision control software for our software repository. We run CVS as a 'pserver' on one of our Solaris workstations. Thus we can directly check in and out pieces of software over the network directly. We currently run a simple implicit heat transfer and scalar advection problems (discussed in Section 5) as test problems before committing software modifications to our CVS repository.

Finally, our software design approach for this project began with graphical design using a Unified Modeling Language design tool called GDPro by Advanced Software Technologies. We found it helpful to use the UML class hierarchy diagrams to map out our ideas on software structure. Once we had a graphical design, we were able to generate stubs for our classes automatically using GDPro's forward engineering feature.

4.3. File IO Package

The lowest level package used in CartaBlanca was imported for basic file input and output (IO) from S. J. Chapman, [4]. This package contains classes with methods that enable the developer to write C-language-syntax file print and read statements. These methods are used in the mesh package classes for reading text-based mesh files (see Section 4.6) and in the graphics package classes for writing text-based graphics files (see Section 4.10).

4.4. Input Package

The input package contains the basic input facilities for problem specification. The user specifies parameters such as solver tolerances, physical properties and boundary conditions using a graphical user interface (GUI). Problem data is written to a 'ProblemSpecifier' class object. The 'ProblemSpecifier' object is a simple Java Bean, [6], that contains all problem specifications from the GUI and can be queried by other objects as needed. This class of objects is serializable. This feature is used to save ProblemSpecifier settings to disk. This eliminates the need for any text-based input files, other than the mesh files (see section 4.6).

The GUI is contained in three classes named 'TabbedInputClass', 'TabbedInputFrame' and 'TabbedInputFrame_AboutBox.' The GUI covers several categories of input separated into several tabbed input frames. The input categories are General Information, Physics, Linear Solver, Non-Linear Solver, Pre-conditioner, Initial Conditions,

Boundary Conditions and Materials. A snapshot of the GUI interface is shown in Figure 2.

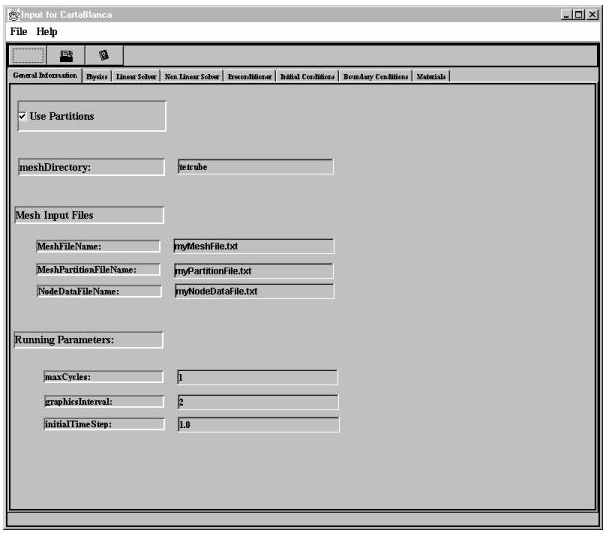

Figure 2. Snapshot of GUI. Tabs enable user to provide input on the various categories of input.

The user can click on the various tabs along the top of the GUI to access the different categories of input. As the user types in new information into the fields of the GUI, the information is written to the ProblemSpecifier object. When the user exits the GUI, the ProblemSpecifier is output to disk as a serialized object for future use and the rest of the program then begins executing based on the information in the ProblemSpecifier object.

4.5. Communications Package

The communications package contains classes of objects that provide functionality for inter-partition communication and for global mesh operations. The class CyclicBarrier provides a simple barrier that objects may invoke to synchronize calculations. The implementation was modeled on the barrier class provided in Chapter 5 of Oaks and Wong, [16]. The CyclicBarrier is used, for example, in discrete operations such as divergence field computations in which communication of flux quantities among mesh partitions are required.

The Reduction class in the communications package provides for the computation of global quantities across the entire mesh such as a global maximum or a global sum. Global sums are required, for example, for mesh-wide dot products of vectors in the various Krylov solvers. The Reduction class accomplishes this by using static class variables for sums and extrema.

4.6. Mesh Package

The mesh package contains several classes that describe mesh elements, edges, interior boundary nodes, and partition and global meshes. Let us discuss these classes in the order in which they come into existence as the program reads in mesh information from mesh files. To begin, let us describe the three types of mesh information files that CartaBlanca requires. The mesh file format follows from those required by the Metis mesh-partitioning program, [13]. The three files contain the mesh connectivity, the node coordinates and the partitioning of the mesh elements. Please see the Metis manual, [13], for a description of these files.

CartaBlanca requires mesh partitioning to be done in such a way that elements and not nodes are partitioned. Referring to Figure 3, the mesh partitioning for CartaBlanca must be done along node-edge connections. In the Figure, the heavier edge connections denote the boundary between partition A and partition B. To implement this mode of partitioning in CartaBlanca, nodes on the partition boundaries are duplicated. In the example in the Figure, the three nodes along the partition boundary would be present in each partition as duplicates.

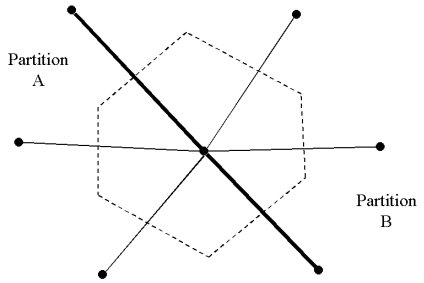

Figure 3. Partitioning in CartaBlanca. Meshes must be partitioned along node connections.

To illustrate further how mesh partitioning works in CartaBlanca, a two-dimensional mesh is shown in Figure 4

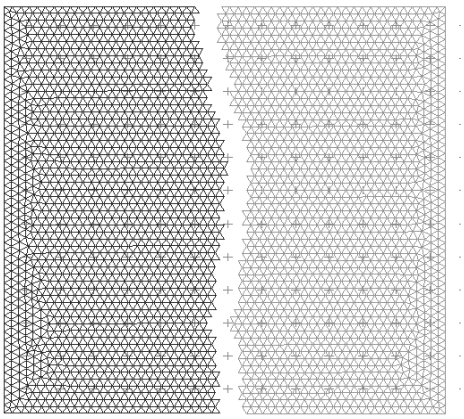

Figure 4. Two-dimensional partitioned mesh.

The mesh partitioning shown in Figure 4 was performed using the Metis program and the Metis output was then fed to CartaBlanca for computations. The actual plot was generated using the Tecplot program which operates on graphics output files from CartaBlanca (see Section 4.10) A further example mesh is shown in Figure 5 for the case of a 3-dimensional tetrahedral mesh.

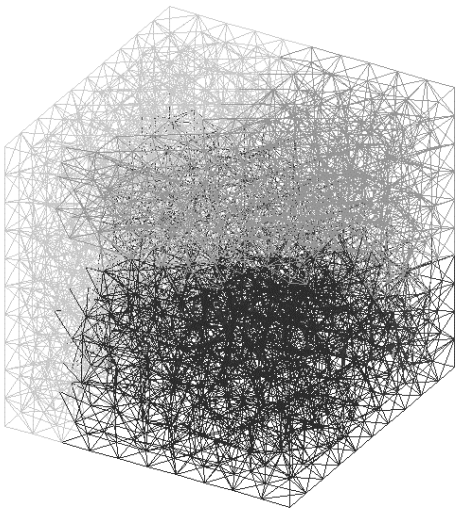

Figure 5. Three-dimensional tetrahedral element mesh. The shading denotes the 4 partitions that were computed by Metis.

In general terms, the mesh package classes perform the following functions:

- Read and store data from mesh input files. This includes element connectivity, node coordinates and element partitioning,
- Compute all required element and node geometrical information such as cell face areas and normal vectors for the global mesh,
- Compute all edge connectivity and geometric information from the element information for the global mesh,
- Link all nodes via edge elements,
- Setup up partition meshes with links between global and partition mesh objects including nodes, elements and edges.
- Set up connectivity between duplicate nodes on different partitions.

4.7. Discrete Operations Package

The discrete operations package contains a class called Divergence which provides a variety of mesh-wide discrete operations including the computation of the divergence of a vector field, the gradient of a scalar at both mesh nodes and mesh faces as well as some more specialized operations. Some of the specialized operations include finding the maximum face-by-face inflow values for each node for advection calculations and finding the diagonal term of a mesh-wide matrix operator. All of these operations require communication and therefore use the duplicate node connectivity information from the mesh package classes and the barrier object from the communications package.

4.8. Physics Package

The physics package contains classes that allow a developer to input and specify the conservation equations that he or she would like to solve. The developer first must set up an AbsState class corresponding to his physical system. This is a container class that is discussed below in section 4.8.1. Once the AbsSstate class is set up, the user then can specify his conservation equations in an AbsProblemPhysics class. This is discussed in section 4.8.2. When specifying both the AbsState class and the AbsProblemPhysics class, the user must extend abstract classes that provide the basic format that is expected by the rest of CartaBlanca.

4.8.1. AbsState Class

The AbsState class is an abstract class that must be extended by the developer to provide a data container for state variables for specific physics problems. The state variables are fundamentally stored in a two dimensional array wherein the first dimension is the variable type and the second dimension is the node index. So, for example, if one is trying to solve a problem with state variables for pressure, and three components of velocity, then the first dimension of this array would be four. The two-dimensional representation is convenient for developers since they tend to work with the governing equations a field or state variable type at a time. The two-dimensional view is also a convenient format for the graphics package since it also processes the data a field at a time.

Krylov solvers, however, work in terms of a one-dimensional state vector. Thus, the AbsState class also provides a one-dimensional view of the same state data. Currently, the one-dimensional view is provided as a copy of the two dimensional data. The copy is performed using Java's System.arraycopy function for best performance. When and if Java provides a true two-dimensional array, [11], this copy may be avoided altogether.

4.8.2. AbsProblemPhysics Class

For linear physical systems, developers can specify their physical system behavior by extending the AbsProblemPhysics class. AbsProblemPhysics is an abstract class that lays out what CartaBlanca expects from physics objects. The most important feature of this class of objects is the methods to get the right and left hand side of the governing equations for the state variables. The solvers in CartaBlanca interact with these physics object methods to obtain the right-hand side of the linear equation system and the matrix-vector multiply. Another important behavior of AbsProblemPhysics objects is the pre-conditioning method. The Krylov solvers also interact with physics objects by invoking their pre-conditioning method. This method takes a Krylov vector from the Krylov solver and updates it according to some iterative improvement scheme. Currently, a Jacobi iteration scheme is used. Plans for a multigrid scheme are in place to obtain improved solver performance.

AbsProblemPhysics classes also inherit some methods for the base classes for converting time n states to time $n+1$ states. These methods can, of course, be overridden in the derived classes to provide additional functionality.

4.8.3. NLAbsProblemPhysics Class

In the case of nonlinear physics problems, the matrix-vector multiply evaluation has to be provided in a generic fashion following Equation (9). The NLAbsProblemPhysics class of CartaBlanca extends the AbsProblemPhysics to provide this behavior. In this class of objects, the developer must encode the governing equations into methods that return the full nonlinear residual equation in the form of a left and right hand side. The left and right hand side correspond to the implicit and explicit parts of the governing equations. These objects invoke these nonlinear *get* methods from the overridden linear *get* methods

from AbsProblemPhysics class using Equation (9) to produce a linear matrix-vector multiply evaluation. Since NLAbsProblemPhysics inherits from AbsProblemPhysics, all other behavior, such as pre-conditioning is available.

Figure 6 provides a graphical overview of the physics class inheritance hierarchy that was generated directly from the Java source code using GDPro.

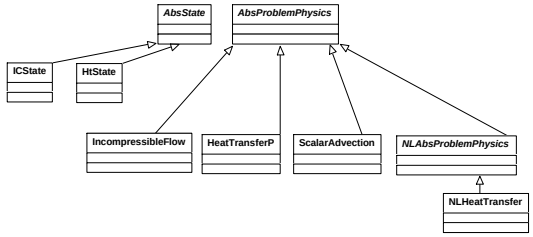

Figure 6. UML Class hierarchy diagram of the Physics package. Note that NLHeatTransfer inherits from NLAbsProblemPhysics which inherits from AbsProblemPhysics.

4.9. Solver Package

The solver package contains classes for linear and nonlinear solvers. As for the classes in the physics package, an abstract solver class, AbsSolver, is provided as a parent for all solvers. Currently, this class has been extended to provide users a Conjugate Gradient and Gmres Krylov solver class. In addition, an 'explicit' solver is provided for fully explicit calculations which essentially bypasses any solution method at all and simply returns the right hand side as the solution. Finally, a Newton-Gmres (JFNK) solver is provided for nonlinear problems. Each of these solvers communicates directly with physics objects through method invocations. The solver class inheritance hierarchy is shown in Figure 7.

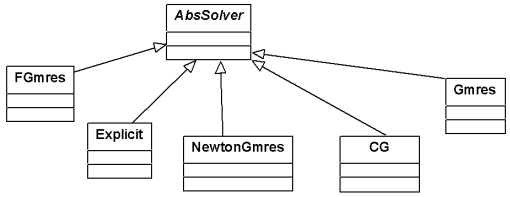

Figure 7. Solver package class hierarchy.

Included in the figure is a class diagram for Fgmres, or Flexible Gmres that allows for increased pre-conditioning options. This will be added at a later date to allow for increased solver method prototyping options.

4.10. Graphics Package

The Graphics package, at present, contains only one class that can be used to produce Tecplot output files. The class interacts with the abstract state class so that it automatically knows about new state variables, etc. As mentioned in Section 4.3, the graphics class currently uses the Chapman IO facility to produce a text-based file. Eventually, this class will be extended to allow for additional plot file output formats. We also envision direct use of Java graphics.

4.11. Problem Driver Package

The ProblemDriver package contains the Driver class, a top-level driver for solving physics problems on each mesh partition. The Driver class implements Java's Thread-class Runnable interface. This enables data-parallel computation in CartaBlanca with each thread corresponding to a particular mesh partition. Figure 8 shows a UML association diagram for the Driver class.

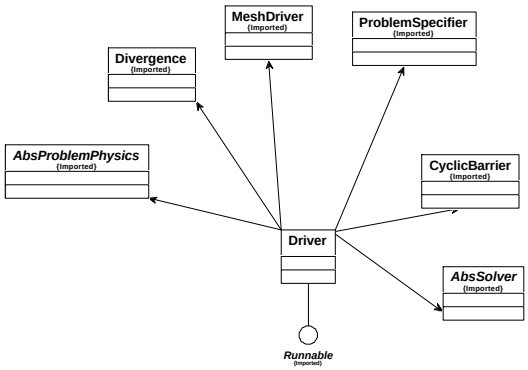

Figure 8. Association diagram for driver class.

As can be seen, the Driver class interacts with all the major CartaBlanca objects from an AbsProblemPhysics object to an AbsSolver object.

4.12. Main Package

The main package consists of several classes that contain the public static main method that drives the entire simulation. The class PhysMain contains a main method that instantiates all high-level objects and invokes the start method for all of the Driver objects for each mesh partition.

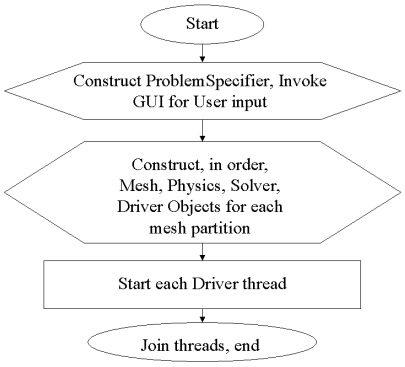

Figure 9. Flow chart for CartaBlanca main method.

5. RESULTS

Here we provide information on compilation, run-time performance and three preliminary simulation results from CartaBlanca. This work was performed using an SGI 320 Dual Processor workstation with 500 MHz Pentium III processors and the Windows NT operating system. All of the physics simulations were performed using two threads for two-processor data parallel computations.

5.1. Compilation

The entire software package compiles to byte code using JBuilder 4.0 with the Java 1.3 JDK in a matter of a few seconds. In addition, we have begun to use the optimizing compiler called JOVE (version 2.0) by Instantiations, Inc to produce native code executables of CartaBlanca. Without the GUI, the code compiles from byte code to native code executable in a matter of 3 to 4 minutes. With our GUI, the compilation takes about an hour, however.

5.2. Scalar Advection

The first two examples are of explicit scalar advection calculations done on the triangular element grid shown in Figure 4. The results of the calculations are shown in Figure 10 and Figure 11. These calculations were performed using 2 threads in parallel on the two-processor workstation. Each thread operated on a separate mesh partition.

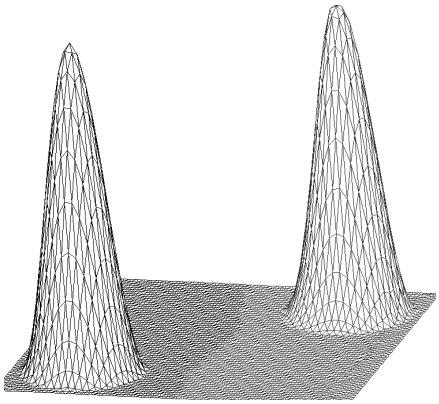

Figure 10. Results from a continuum scalar advection simulation on a two-dimensional triangular mesh.

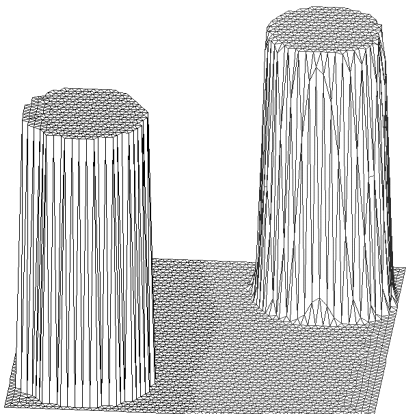

Figure 11. Results from a interface-tracking scalar advection simulation on a two-dimensional triangular mesh.

Each calculation started with a pulse of concentration in the lower left hand corner of the domain. In the first example, the concentration pulse has a Gaussian spatial distribution while in the second example; the pulse was a spatial step function with circular shape. This material was advected towards the top right according to the conservation equation

$$\frac{\partial c}{\partial t} + \frac{\partial c u_i}{\partial x_i} = 0 \qquad (10)$$

where c is concentration, u_i is the ith component of velocity, t is time and x_i is the ith spatial coordinate. There is an implied summation over the repeated index in the second term. In these example calculations, the velocity was a constant with component values of one in the horizontal and vertical directions. The final conditions of the concentration pulses are shown in the top-right of Figure 10 and Figure 11. Note that in the case of Figure 10 we used the continuum advection version of CartaBlanca's advection facility while in Figure 11 we used the interface-tracking version of the advection facility.

The calculations used Cartablanca's explicit solver to advance the solution of Equation (10) over 120 time cycles. Although Equation (10) is relatively simple, it is commonly known that it is difficult to obtain accurate numerical solutions for this equation that minimize the artificial numerical smearing of such concentration pulses and also avoid the creation of artificial extrema, especially on triangular grid meshes. Also, doing such calculations using interface tracking is also a very difficult task. These examples demonstrate that CartaBlanca has advanced advection algorithms that can be used by developers for a wide variety of applications.

5.3. Implicit Heat Transfer

The second example is a simple implicit heat transfer calculation performed on the same mesh. CartaBlanca's Gmres solver was used to compute the temperature field as a function of time over 5 time cycles according to the equation

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x_i}\left(a \frac{\partial T}{\partial x_i} \right) \qquad (11)$$

where T is temperature and a is the thermal diffusivity, which was set to one for the example problem. Zero-gradient boundary conditions were used which correspond physically to a perfectly insulated box. The final state is, therefore, a uniform temperature of one-half. CartaBlanca found this solution, as expected, using both a single mesh partition and with the 2 mesh partitions on the SGI 2-processor workstation mentioned above. The observed speed-up going from one to two processors was a factor of about 1.5. While this is not perfect scaling, our experience from other parallel codes has been that good scaling is observed only with more than 4 processors. More work is required to completely understand the scaling characteristics of CartaBlanca.

5.4. Performance

While we are still in the infrastructure-building phase of this project and have not yet been able to spend time optimizing code, we can provide some preliminary indications of performance. In order to compare with C++, for example, we have timed the computation of the divergence of a vector field on an unstructured mesh using both CartaBlanca and a C++ version of the same algorithm. (with edge-based indirect addressing). The C++ version of the algorithm was from the Kokopelli code mentioned in Section 1.3, which was compiled with optimization. We found

that the byte-code-interpreted Java run time was about the same as the C++ run time. While we expected some performance increase with the native-code compiled version of CartaBlanca using the JOVE product, we, unfortunately, saw speed-ups of only a few percent. We need to spend more time to fully understand these preliminary results.

6. CONCLUSIONS AND FUTURE PLANS

Since we are still very much in the development phase of this project, it is too early to provide definitive assessments of the design and performance of CartaBlanca. In a qualitative sense, we can say that CartaBlanca has already been a useful tool for algorithm development since we have been able to do the research on our new advection scheme exclusively within CartaBlanca. That is, CartaBlanca has provided us with sufficient usability and performance that we did not choose to go "off-line" to some other program or development tool to do one of our main jobs, algorithm research and development. So in this sense, we can already claim some "bottom-line" success.

In the future, we plan to pursue a number of promising leads in the near future. The following is a partial list covering some of the major possible directions:

- Implement a fully coupled multiphase flow capability for weapons-complex and other applications. This would put us in a position to examine a number of important flow simulation problem classes including single-phase flow, interfacial flows and potentially, fluid-structure interactions.
- Implement and investigate operator-split pre-conditioning strategies for complex coupled systems. This should provide significant speed-up to our Jacobian-Free Newton-Krylov solver facility. Once this is fully in place, we should be able to better assess how well CartaBlanca and Java could serve as a serious production simulation tool.
- Implement a moving and adaptive mesh capability. This capability would further the utility of CartaBlanca and may be particularly important for fluid-structure interaction simulation problems. Furthermore, it would test how well we have encapsulation the present meshDriver object when we replace it with one that can perform mesh motion.
- Fully investigate the use of Java native-code compilers.
- Extend CartaBlanca to distributed memory architecture for cluster-based computing. We are considering the use of JavaParty, [19], as a means to extend CartaBlanca to distributed memory systems.

7. ACKNOWLEDGMENTS

We gratefully acknowledge the support for this work from the Department of Energy and the Los Alamos Computer Science Institute (LACSI). We thank Dana Knoll, Doug Kothe and Manjit Sahota for their very useful comments and suggestions. We also thank John Thorp, Joel Dendy and Stephen Lee for their programmatic advocacy on our behalf.

8. REFERENCES

[1] F. L. Addessio, J. R. Baumgardner, J. K. Dukowicz, N. L. Johnson, B. A. Kashiwa, R. M. Rauenzahn, C. Zemach, *CAVEAT: A Computer Code for Fluid Dynamics Problems with Large Distortion and Internal Slip, Los Alamos National Laboratory Report LA-10613-MS, Rev. 1*, May, 1992.

[2] R. M. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, 1994.

[3] P. N. Brown and Y. Saad, Hybrid Krylov Methods for nonlinear systems of equations, *SIAM J. Sci. Stat. Comput.*, 11(3):450-81, 1990.

[4] S. J. Chapman, *Java for Engineers and Scientists*, Prentice Hall, Upper Saddle River, NJ, 2000.

[5] R. M. Eckstein, M. Loy and D. Wood, *Java Swing*, O'Reilly, Cambridge, 1998.

[6] R. Englander, *Java Beans*, O'Reilly, Cambridge, 1997.

[7] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*, Springer, New York, 1999.

[8] M. Fowler, Put Your Process on a Diet, *Software Development Magazine*, 2(12), December 2000.

[9] M. Hatakeyama, M. Watanabe and T. Suzuki, Object-Oriented Fluid Flow Simulation System, Computers & Fluids, 27(5):581-597, 1998.

[10] J. Hauser, T. Ludewig, T. Gollnick, R. Winkelman, R. Williams, J. Muylaert and M. Spel, A Pure Java Parallel Flow Solver, *AIAA paper 99-0549.*

[11] Java Grande Forum Panel, *Java Grande Forum Report: Making Java Work for High End Computing*, SC 1998, Orlando, Florida, 1998.

[12] JUnit, http://www.JUnit.org/.

[13] G. Karypis, and V. Kumar, *METIS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*, University of Minnesota, Department of Computer Science / Army HPC Research Center, Minneapolis, MN (http://www-users.cs.umn.edu/~karypis/metis/index.html)

[14] B. Kashiwa, N. T. Padial, R. M. Rauenzahn and W. B. VanderHeyden, A Cell-Centered ICE Method for Multiphase Flow Simulations, *FED-Vol. 185, Numerical Methods in Multiphase Flows, ASME*, 185:159-176, 1994.

[15] C. T. Kelly, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995.

[16] S. Oaks, and H. Wong, *Java Threads*, O'Reilly, Cambridge, 1999.

[17] J. V.W. Reynders III and J. Cummings, The POOMA Framework, *Computers in Physics*, 12(5):453-459, 1998.

[18] P. J. O'Rourke, and M. S. Sahota, *CHAD: A Parallel, 3-D. Implicit, Unstructured-Grid, Multimateria, Hydrodynamics Code for All Flow Speeds*, Los Alamos National Laboratory Report LA-UR-98-5663, October, 1998.

[19] M. Philippsen, and M. Zenger, JavaParty: transparent remote objects in Java, *Concurrency-Practice and Experience*, 9:1225-1242, November 1997.

[20] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, San Fransisco, 1995.

[21] J. Schatzman and R. Donehower, High-Performance Java Software Development, *Java Report*, 6(2):24-41, 2001.

[22] V. Selmin, The Node-Centered Finite Volume Approach: Bridge between Finite Differences and Finite Elements, *Comput. Methods in Appl. Mech. Engrg.*, 102(1):107-138, January, 1993.

[23] Telluride, http://public.lanl.gov/mww/HomePage.html, Los Alamos National Laboratory Report LA-UR-99-1664, 1999.